AWK 教程

half-beast@163.com

Jun. 1 , 2015

1 AWK 教程

Awk 是一门方便其表达力丰富的编程语言，它适用于多变量计算和数据操作任务。本章是一个教程，目标是使你能够尽快的用 awk 写出属于自己的程序。第二章详细描述了语言的方方面面，而剩下的章节则向你展示 awk 习惯于解决许多领域的问题。整本书中，我们都试图挑选一些让你感觉既有意思同时又很有启发性的例子。

1.1 开始

有用的 awk 程序通常都很短，有的甚至只有一到两行。假设你有一份称为 emp.data 的文件。它包含雇员名字，时新和雇员的工作时常，一条雇员记录单独成行，如下所示：

Beth 4.00 0
Dan 3.75 0
Kathy 4.00 10
Mark 5.00 20
Mary 5.50 22
Susie 4.25 18

现在你想打印所有那些工作时间超过零小时的雇员名称和个人所得。awk 天生擅长这类工作，对 awk 来说，这太容易了。只需敲入以下命令行：

awk ‘$3 > 0 { print $1 , $2 * $3 }’ emp.data

你应该得到如下输出：

Kathy 40
Mark 100
Mary 110
Susie 76.5

这条命令告诉系统使用 awk 来执行单引号内的程序，并从输入文件 emp.data 中获取处理数据。单引号中的内容是一条完整的 awk 程序。它由单条 pattern-action 语句构成。pattern–$3 > 0，用于匹配每一输入行的第三列或字段是否大于 0，对应的 action–{ print $1 , $2 * $3 } 打印匹配行的第一列和第二列与第三列的乘积。

如果你想打印那些没有工作的雇员名称，敲入如下命令：
awk ’$3 == 0 { print $1 }’ emp.data
此处 pattern –$3 == 0，匹配每行数据的第三个字段是否等于 0，对应的 action–{ print $1 }，打印匹配行的第一个字段。

当你读这本书的时候，最好试着去执行和修改上述的程序。因为大部分程序都很短，所以你能够很快的明白 awk 工作的原来。在 Unix 系统上，上述的两个例子在终端显示如下：
$awk ’$3 > 0 { print $1 , $2 * $3 } ’ emp.data
Kathy 40
Mark 100
Mary 110
Susie 76.5
$awk ’$3 == 0 { print $1 } emp.data
Beth
Dan
$
每行的起始符号 $ 是 Unix 系统的输入提示符；它可能与你的系统有所区别。

Awk 程序的结构
让我们退后一步看看到底发生了什么事情。上面命令行中的单引号部分是用 awk 语言写的程序。在这一章出现的所以 awk 程序都是由单条或者多条 patter-action 语句组成的：
pattern { action }
pattern { action }
...
awk 程序最基本的操作是逐条扫描输入行，以搜索到那些与 pattern 相匹配的行。在讨论中，”匹配（match）”一词的确切意思依赖于 pattern；例如：如果 pattern 是 $3>0，那么它的意思是“此条件为真”。
所有的 pattern 都会对输入的每行数据依次进行测试。只要有一个 pattern 匹配上，则相关的 action 就会被执行。然后，读入下一行，重新开始匹配。直到处理完所有的行，程序才结束。以上的程序是典型的 pattern 和 action 的例子。
$3 == 0 {print$1} 是单个 pattern-action 语句；只要哪行的第三字段等于 0，那么就打印该行的第一字段。
可以省略 pattern-action 语句中的 pattern 或者 action（但不同时）。如果 pattern 没有对应的 action，例如：$3 == 0 ，那么匹配的行（即，$3==0 返回真的行）都会被打印出来。当输入文件是 emp.data 时，该程序会打印第三字段为 0 的两行：
Beth 4.00 0
Dan 3,75 0
如果 action 没有对应的 pattern，例如：{print$1 }，那么该 action 会每行数据的第一个字段。

因为 pattern 和 action 都是可以选择的，所以 action 用花括号括起来，以与 pattern 相区别。

执行一个 awk 程序

有许多方式来执行一个 awk 程序。你可以在命令行下输入 awk 'program' input files 来对买个输入文件执行'program'。例如，你可以输入

awk '$3 == 0 { print $1 } ' file1 file2

来打印文件 file1 和 file2 中第三个字段是 0 的所有行。

你可以在命令行中省略输入文件，而仅仅输入

awk 'program'

在这种情况下，awk 会对你在终端的任何输入执行 program，直到你键入文件结束信号（ctr-d 在 Unix 系统下）。下面是在 Unix 系统下的一个简单会话：

$ awk ' $3 == 0 { print $1 } '
Beth 4.00 0
Beth
Dan 3.75 0
Dan
Kathy 3.75 10
Kathy 3.75 0
Kathy
...

加粗的字符串是计算机打印的。

此种方式很容易用 awk 做实验：输入你的程序，然后输入数据，接着查看执行结果。我们再次鼓励你尝试输入这些例子，也可以尝试修改例子。

注意在命令行中程序要用单引号引起来。这样做是为了防止程序中的字符 $ 被 shell 解释，并且可以允许程序超过单行限制。

当程序很短时（只有几行），此种执行方式很方便。但是，当程序很长的时候，把程序写入单独的文件（如：progfile），则是更方便的做法。输入如下命令行：

awk -f progfile optional list of input files

参数 -f 告诉 awk 根据文件名获取程序。可以用任何文件名称来代替 progfile。

错误

如果你在 awk 程序中犯了一个错误，那么 awk 将输出一个诊断信息。例如，如果你错误输入括号，如下：

awk '$3 == 0 { print $1 } ' emp.data

你讲得到如下提示信息：

awk : syntax error at source line 1
context is
$3 == 0 ⋙ [⋘
extra }
missing }

awk :bailing out at source line 1

”Syntax error“意味着你在”⋙⋘“标注的地方犯了一个语法错误。”Bailing out“意味着恢复尝试失败。有时候，你可能获得一个更有用的信息，比如错误的大括号或圆括号匹配信息。

由于语法错误，awk 将不会去尝试执行程序。然而，有些错误只有在运行的时候才可能被检测出来。例如，如果你尝试用 0 除一个数，那么 akw 将立即停止程序，并报告输入的行号和程序错误的位置。

1.2 简单输出

剩下的章节包含许多短小经典的 awk 程序，主要用于处理以上章节提到的 emp.data 文件。我们将简短的介绍接下来的内容，但是这些例子同时也说明了 awk 很擅长这方面的操作 –打印字段，过滤输入和转换数据。我们不打算展示 awk 所有的功能，但是我们会针对特殊情况进行细节上的讨论。学完本章，你将会完成相当多的例题，最后你会发现能够更容易的理解接下来的内容。

我们将会只展示程序部分，而不是整个命令行。无论是把程序用单引号引起来作为命令行的一部分执行，还是把程序写进文件再以 -f 参数的形式调用，这些程序都能够正常运行。

awk 语言仅仅只有两种数据类型：数字和字符串。emp.data 文件具有这两种类型的典型特征 –字符串组合和被空格或制表符分隔的数字。

Awk 每次只读入输入内容的一行数据，同时把该行分隔成多个字段，默认情况下，每个字段是一个不含空格或制表符的字符序列。当前行的第一个字段用 $1 表示，第二个字段用 $2 表示，依次类推。整行数据用 $0 表示。字段标号根据具体行改变。

我们通常需要输出每行的部分或者所有字段，或许会进行一些计算。本部分的程序基本都是都是这种样式。

打印所有行

如果 action 没有对应的 pattern，那么 action 会对所有的行进行处理。语句 print 将打印当前所有输入行，所以程序 {print} 打印标准输入的所有行。因为 $0 代表整行，所以 { print $0 } 与 { print } 操作相同。

打印指定字段

使用单条语句可以在同一输出行打印多个字段。程序 {print $1,$3} 可以打印每一输入行的第一和第三个字段。当输入文件是 emp.data 时，输入如下：

```
Beth 0
Dan 0
Kathy 10
Mark 20
Mary 22
```

```
Susie 18
```

打印语句中被逗号分隔的表达式在输出过程中默认用空格代替。print 会在所有生成语句的末端添加换行符。以上两种默认操作都可以被修改。这些我们将在第二章展示。